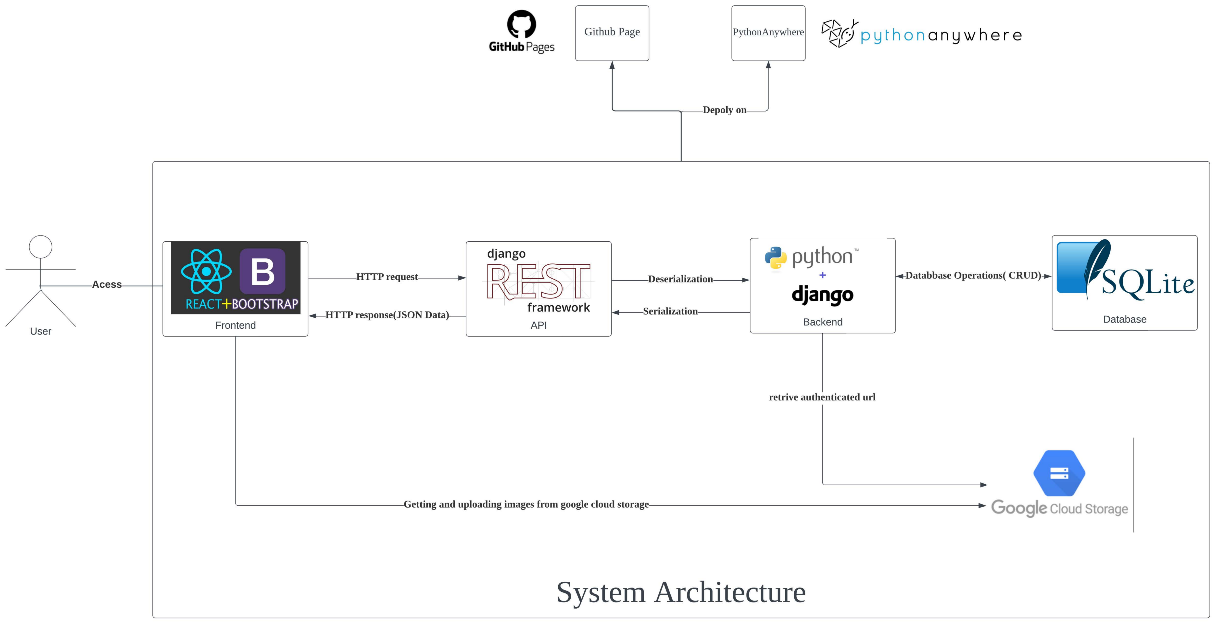
Handover

Introduction

Our product is a state-of-the-art music library software designed for music enthusiasts. With its intuitive interface, users can swiftly add songs to their personalized music playlists, select their favorite images to represent each playlist, and mark their most-loved tracks. Moreover, for businesses or teams, our software offers seamless management capabilities, allowing staff to effortlessly oversee all music lists.

System Architecture



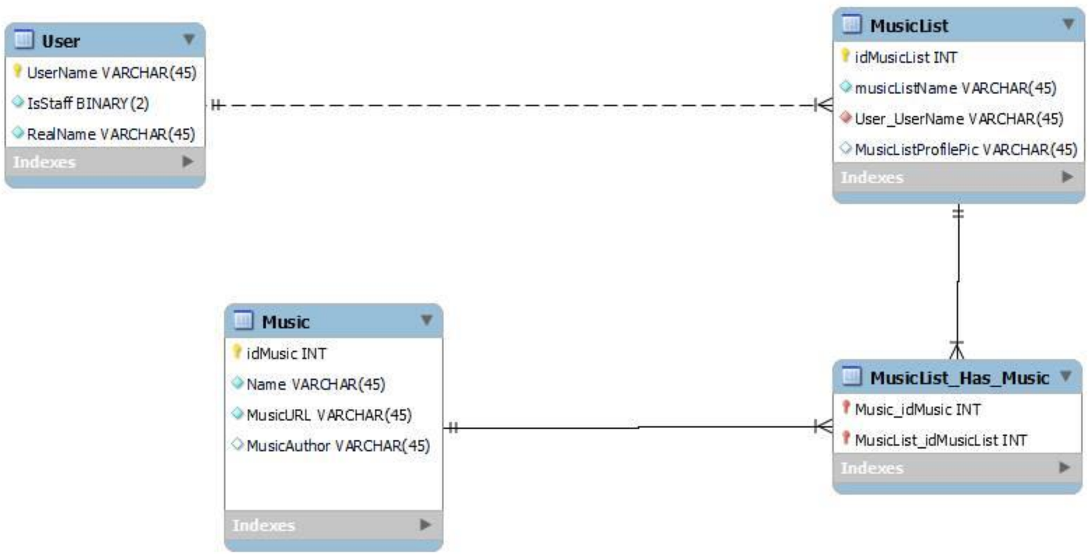
System Architecture

High Level View

- **Front end:**
  - **React Framework:**
    - Uses a component-service structure.
      - **Service File:**
        - ✧ Responsible for communicating with the backend controller.
        - ✧ Acts as an encapsulation layer, ensuring the component file remains unaffected by the environment of HTTP requests and responses.

        - **Component File:**
            - ✧ Receives data fetched by the service file.
            - ✧ Renders the data using HTML and CSS.
            - ✧ If there's a need to modify the database, it initiates POST or PUT requests through the service file.
- **Back end:**
    - **Django Framework:**
        - Adopts a decoupled architecture.
            - ■ **Controller Layer (via Django Rest Framework):**
                - ✧ Handles HTTP requests (GET/POST/PUT/etc).
                - ✧ Translates JSON parameters to objects.
                - ✧ Interacts with the Service Layer.
            - ■ **Service Layer:**
                - ✧ Manages business logic.
                - ✧ Utilizes data provided by the persistence layer.
            - ■ **Persistence Layer:**
                - ✧ Contains storage logic and translates business objects to and from database rows.
                - ✧ Uses Django's integrated SQLite for database operations.
            - ■ **Database (SQLite):**
                - ✧ Where CRUD operations are performed.
                - ✧ Managed through Django's ORM.
- **Storage:**
    - **Google Cloud Drive:**
        - ✧ Securely stores playlist images.
        - ✧ Due to permission constraints, the frontend requests the backend for upload and retrieval links for images.
        - ✧ The backend, utilizing the `google.cloud` library and a locally stored key (`corded-evening-400913-ce5bbb69ae1e.json`), generates links for uploading or retrieving images and sends them to the frontend.
        - ✧ The frontend then makes requests directly to Google Storage using these links.

Database Structure



Setups before run the product

- Download node.js.
- install dependencies. (npm install)
- Download Python3.9 or Python3.10.
- Create a virtual environment.
- install dependencies. (pip install -r requirements.txt)

Run the product

- Run front end Server:
  - npm start
- Run back end Server:
  - python manage.py runsever

For specific instructions, please refer to the readme.md.

Coding Standards

- Comments: Use comments to explain the purpose of complex or non-intuitive code blocks.
- Indentation: Use 4 spaces for indentation.
- Naming Conventions: Use meaningful and descriptive variable, function, and class

names.

For detailed information, please refer to the CoodingStandards section in Confluence.

API quick guide

- Brief overview of the purpose and functionality of the API.
- Backend url: http://127.0.0.1:8000.

Music List API:

- Retrieve all music lists for a specific user.
- Add a new music to a list.
- Update the picture of a music list.
- Remove a music from a list.
- Upload a file.

Music API:

- Fetch all music entries.
- Add a new music entry.
- Update an existing music entry.
- Delete a specific music entry.

User Role Management API:

- Retrieve all user roles.
- Add a new user role.
- Update an existing user's username.
- Delete a specific user role.
- User login functionality.

Response Messages:

- The API provides specific response messages for different actions, such as successful addition, updates, deletions, and login attempts. It also handles errors like username conflicts and invalid inputs.

Status Codes:

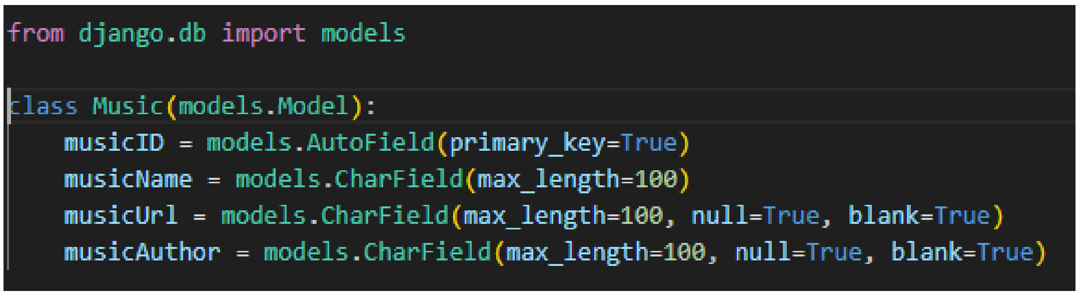
- The API returns specific HTTP status codes based on the outcome of the request, such as 201 for successful operations and various 400-level codes for different types of errors.

For a detailed guide and specific instructions on how to use each endpoint, please refer to the API Quick Guide on Confluence.

Database quick guide

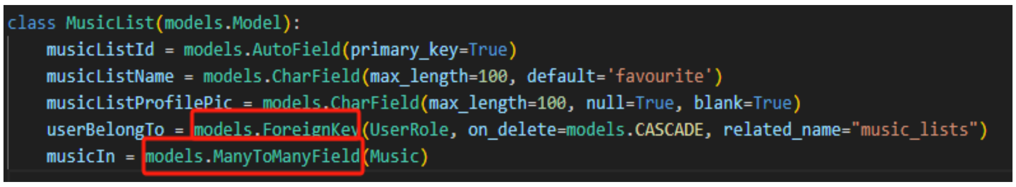
Django's integration with SQLite offers a seamless and efficient way to manage databases for web applications. The convenience lies in Django's ORM (Object-Relational Mapping) system, which abstracts the database operations, allowing developers to focus on the application logic rather than SQL intricacies.

- Direct Model-to-Table Mapping
  In Django, defining a model in your application directly translates to creating a table in the SQLite database. Each attribute in the model corresponds to a column in the table. For instance:



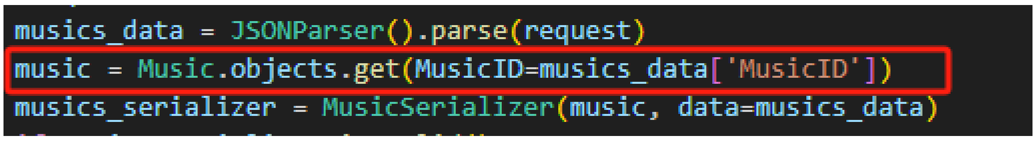
```python
from django.db import models

class Music(models.Model):
    musicID = models.AutoField(primary_key=True)
    musicName = models.CharField(max_length=100)
    musicUrl = models.CharField(max_length=100, null=True, blank=True)
    musicAuthor = models.CharField(max_length=100, null=True, blank=True)
```

- Relationships Between Tables
  Django provides intuitive ways to define relationships between tables:
  - ForeignKey: Represents a many-to-one relationship.
  - ManyToManyField: As the name suggests, it's used for many-to-many relationships between tables.
  - OneToOneField: For one-to-one relationships.



```python
class MusicList(models.Model):
    musicListId = models.AutoField(primary_key=True)
    musicListName = models.CharField(max_length=100, default='favourite')
    musicListProfilePic = models.CharField(max_length=100, null=True, blank=True)
    userBelongTo = models.ForeignKey(UserRole, on_delete=models.CASCADE, related_name="music_lists")
    musicIn = models.ManyToManyField(Music)
```

- Querying the Database
  Django's ORM allows for querying the SQLite database without writing raw SQL. For example, to fetch all songs by a particular artist:



```python
musics_data = JSONParser().parse(request)
music = Music.objects.get(MusicID=musics_data['MusicID'])
musics_serializer = MusicSerializer(music, data=musics_data)
```

- Migrations
  Once models are defined, you can easily create or modify the database structure using Django's migration system

1. Run the makemigrations command

```
python manage.py makemigrations
```

2. Execute the migrate command

```
python manage.py migrate
```